Trie

May 5, 2019

1 Building a Trie in Python

Before we start let us reiterate the key components of a Trie or Prefix Tree. A trie is a tree-like data structure that stores a dynamic set of strings. Tries are commonly used to facilitate operations like predictive text or autocomplete features on mobile phones or web search.

Before we move into the autocomplete function we need to create a working trie for storing strings. We will create two classes: * A `Trie` class that contains the root node (empty string) * A `TrieNode` class that exposes the general functionality of the Trie, like inserting a word or finding the node which represents a prefix.

Give it a try by implementing the `TrieNode` and `Trie` classes below!

```
In [1]: ## Represents a single node in the Trie
        class TrieNode:
            def __init__(self, val=''):
                ## Initialize this node in the Trie
                self.value = val
                self.next = {}

            def insert(self, char):
                ## Add a child node in this Trie
                self.next[char] = self.next.get(char, TrieNode(char))


        ## The Trie itself containing the root node and insert/find functions
        class Trie:
            def __init__(self):
                ## Initialize this Trie (add a root node)
                self.root = {}

            def insert(self, word):
                ## Add a word to the Trie
                #print(self.root)
                #print(word[0])
                word = word
                self.root[word[0]] = self.root.get(word[0], TrieNode(word[0]))
                current = self.root[word[0]]
                for i in range(1, len(word)):
```

```
            current.insert(word[i])
            current = current.next[word[i]]

    def find(self, prefix):
        ## Find the Trie node that represents this prefix
        if not prefix:
            return self.root

        if prefix[0] not in self.root:
            return None

        current = self.root[prefix[0]]
        if current:
            for i in range(1,len(prefix)):
                if prefix[i] not in current.next:
                    return None
                current = current.next[prefix[i]]
        return current
```

2 Finding Suffixes

Now that we have a functioning Trie, we need to add the ability to list suffixes to implement our autocomplete feature. To do that, we need to implement a new function on the `TrieNode` object that will return all complete word suffixes that exist below it in the trie. For example, if our Trie contains the words `["fun", "function", "factory"]` and we ask for suffixes from the `f` node, we would expect to receive `["un", "unction", "actory"]` back from `node.suffixes()`.

Using the code you wrote for the `TrieNode` above, try to add the suffixes function below. (Hint: recurse down the trie, collecting suffixes as you go.)

```
In [2]: class TrieNode:
            def __init__(self, val=''):
                ## Initialize this node in the Trie
                self.value = val
                self.next = {}

            def insert(self, char):
                ## Add a child node in this Trie
                self.next[char] = self.next.get(char, TrieNode(char))

            def suffixes(self, suffix = ''):
                ## Recursive function that collects the suffix for
                ## all complete words below this point
                #print(self.value)
                #print(self.next)
                #print(suffix, 'suffix', self.next)
                suffix_list = []
                for values in self.next:
```

```
                #print(values, 'val')
                if self.next[values].next:
                    suffix_list.extend(self.next[values].suffixes(suffix+values))
                elif values == '\x00':
                    suffix_list.append(suffix)
            return suffix_list
```

3 Testing it all out

Run the following code to add some words to your trie and then use the interactive search box to see what your code returns.

```
In [4]: MyTrie = Trie()
        wordList = [
            "ant", "anthology", "antagonist", "antonym",
            "fun", "function", "factory",
            "trie", "trigger", "trigonometry", "tripod"
        ]

        for word in wordList:
            MyTrie.insert(word+'\0')
            #print(MyTrie.root[word[0]].suffixes())
        #print(MyTrie.root['f'].next['u'].next['n'].next)

In [5]: from ipywidgets import widgets
        from IPython.display import display
        from ipywidgets import interact
        def f(prefix):
            if prefix != '':
                prefixNode = MyTrie.find(prefix)
                if prefixNode:
                    print('\n'.join(prefixNode.suffixes()))
                else:
                    print(prefix + " not found")
            else:
                print('')
        interact(f,prefix='');
```

```
interactive(children=(Text(value='', description='prefix'), Output()), _dom_classes=('widget-int
```